Churn reduction using R and Python

By

Subhra Kanchan Pattnaik

Submitted on Date: January 25 2019

Contents

Introduction

1.1 Problem Statement

Prediction of the customer behavior. This problem statement is targeted at enabling churn reduction using analytics concept. Since now a days customers are leaving the existing companies and acquiring new companies so the basic need is to predict the customer behavior depend on which we can increase the quality and quantity for the customers so that the existing customers do not leave the existing company. As acquiring new customers is a tedious task and it also increases the competition as acquiring a new customer needs more money and time to be spent.

1.2 Data

Here the public data has been provided based on which we have to create different models on the test data and compare the accuracy of the model in order to predict the churn reduction.

Below are the variables of the data provided for test and train data.

account length ● international plan ● voicemail plan ● number of voicemail messages ● total day minutes used ● day calls made ● total day charge ● total evening minutes ● total evening calls ● total evening charge ● total night minutes ● total night calls ● total night charge ● total international minutes used ● total international calls made ● total international charge

There are two data sets train data and test data so depending on the train data we need to predict the test data.

Chapter 2

Methodology

2.1 Pre Processing

In order to develop a model for a dataset we need to 1st understand the data, structure of data because understanding a data will only help us to identify the variables which are required and which are not required and then we can clean the data and the cleaned data can be used for the final model and predicting purpose. The process of exploring and cleaning of data is often called as **Exploratory Data Analyses**. After the Problem statement is identified then we need to analyze the data and we need to identify the independent and dependent variables also we need to identify the categorical and numerical variable. So these are all the steps of the exploratory Data Analyses also need to identify those variables which are not required. Hence in order to clean the Data and to make the data meaningful we need preprocessing steps. Also we need to check the variable types.

According to the dataset provided train and test data we have 21 independent variables and one dependent variable so we can consider the variable Churn as the dependent variable and start analyzing.

2.1.1 Preprocessing Methods

The 1st preprocessing analyses after doing Exploratory Data Analyses is **Missing Value Analyses.** 1st step is converting the data into a proper data type that is all numeric data and categorical data into a proper data type. Next step is to clean the data and get the proper shape of the data. After applying the missing value analyses we can also apply the **outlier analyses** as well as the **feature selection** and **Feature Scaling** Methods in order to get proper shape of the data on which we can apply different models

2.1.2 Missing Value Analyses

Sometime values are missing in some variables which is called as missing values. It may be due to human error or due to some small cause. If 80% is missing value for a particular variable then we can remove that particular variable and we should not impute the missing fields. According to business it should be less than 30%.
Once we get the missing variables we can use Mean, Median or KNN imputation methods to impute the missing variables.
In this dataset provided for churn reduction we didn’t find any of the missing values and we are moving with Outlier analyses.

2.1.3 Outlier Analyses

Outlier is nothing but the inconsistent values in the dataset. The data may be correct but stands out from other data or is exceptional then it is called outlier analyses. We should remove outliers as outcomes with outlier may give incorrect value which can divert us from wrong prediction.

2.1.4 Feature Selection

Feature Selection is the process of selecting the variables from the provided dataset. We have used correlation and Chi –square test of independence.

2.1.5 Feature Scaling

Normalization and Standardization are the two feature scaling process.

3 Model

We need to develop a model in order to compare the accuracy, false Negative rate and other parameters and depending on that we need to choose the Models.

Below are the models which have been used are :

3.1 Decision Tree Classification Model

3.2 Random Forest Model

3.3 Logistic regression Model

3.4 KNN and Naive Bayes Model

4 R and Python Code

Below is the R code where the data has been converted into proper data type and then we have calculated the missing values.

R-Code:

```
rm(list = ls(all =T))
setwd("E:/python")
x=c("ggplot2", "corrgram", "DMwR", "Caret", "ramdomForest", "unbalanced", "c50",
"dummies", "e1071","Information", "MASS", "rpart", "gbm", "ROSE")
lapply(x, require, character.only = TRUE)
getwd()
train = read.csv("train_data.csv", header = T, na.strings = c("","","NA"))
test = read.csv("test_data.csv", header = T, na.strings = c("","", "NA"))
str(train)
str(test)
#combining the train and test data:
train$training_data = 1
test$training_data = 2
train_test = rbind(train, test)
train_test_backup = train_test
lapply(train_test, FUN = function(x) length(unique(x)))
lapply(train_test, FUN = function(x) sum(is.na(x)))
lapply(train_test, FUN = function(x) sum(is.null(x)))

train_test$Churn = factor(x = train_test$Churn, labels = 0:(length(unique(train_test$Churn))-1))
View(train_test$Churn)
length(unique(train_test$Churn))
unique(train_test$Churn)
train_test$voice.mail.plan = factor(x = train_test$voice.mail.plan, labels =
0:(length(unique(train_test$voice.mail.plan))-1))
View(train_test$voice.mail.plan)
```

```
train_test$international.plan = factor(x = train_test$international.plan, labels = 0:(length(unique(train_test$international.plan))-1))
View(train_test$international.plan)
train_test$phone.number = NULL
View(train_test$phone.number)
train_test$total.day.minutes = NULL
train_test$total.eve.minutes = NULL
train_test$total.night.minutes = NULL
train_test$total.intl.minutes = NULL
View(train_test)
version
# Missing Value Analyses
missing_val = data.frame(apply(train_test, 2, function(x){sum(is.na(x))}))
missing_val$columns = row.names(missing_val)
row.names(missing_val) = NULL
#No missing values found

#Outlier Analyses
numeric_index = sapply(train_test, is.numeric)
numeric_index
numeric_data = train_test[ ,numeric_index]
cnames = colnames(numeric_data)

for(i in 1:length(cnames))
{
 assign(paste0("gn", i), ggplot(aes_string(y = (cnames[i]), x= "Churn"), data = subset(train_test))+
      stat_boxplot(geom = "errorbar", width = 0.5)+
      geom_boxplot(outlier.color="red", fill = "grey", outlier.shape = 18, outlier.size = 1, notch = FALSE) +
      theme(legend.position = "bottom")+
      labs(y=cnames[i],x="Churn")+
      ggtitle(paste("Boxplot of Churn for", cnames[i])))
}
length(cnames)
#Box plot to Vizualize the outliers
gridExtra::grid.arrange(gn1,gn2,gn3,gn4,ncol=4)
gridExtra::grid.arrange(gn5,gn6,gn7,gn8,ncol=4)
gridExtra::grid.arrange(gn9,gn10,gn11,gn12,ncol=4)
gridExtra::grid.arrange(gn13,gn14,ncol=2)
#Time to remove the all the outliers or we can impute it as missing values using boxplot.stat
```

#loop to remove outliers using boxplot

```
for (i in cnames){
  print(i)
  val = train_test[,i][train_test[,i] %in% boxplot.stats(train_test[,i])$out]
  train_test = train_test[which(!train_test[,i] %in% val),]
}
```

#2nd Method: Detect the outlier and replace the outlier with NA

```
for (i in cnames){
  val = train_test[,i][train_test[,i] %in% boxplot.stats(train_test[,i])$out]
  train_test = train_test[which(!train_test[,i] %in% val),]
}
```

#Feature Selection

##correlation

```
corrgram(train_test_backup[,numeric_index], order = F,
        upper.panel = panel.pie, text.panel= panel.txt, main ="correlation plot")
```

##Chi-square test of Independence

```
factor_index = sapply(train_test_backup, is.factor)
factor_data = train_test_backup[,factor_index]
for(i in 1:4)
   {
      print(names(factor_data)[i])
      print(chisq.test(table(factor_data$Churn, factor_data[,i])))
}
```

##Dimension Reduction

```
train_test_deleted = subset(train_test_backup, select = -c())
```

##Feature Scaling

##***Normalization***

```
cnames
for(i in cnames){
  print(i)
  train_test[i] = (train_test[,i] - min(train_test[,i]))/(max(train_test[,i] - min(train_test[,i])))
}
View(train_test_backup)
```

#standardisation

```
for(i in cnames){
  print(i)
  train_test[i] = (train_test[,i] - mean(train_test[,i]))/sd(train_test[,i])
}
```

##Decision Tree Model

```
str(train_test)
View(train_test)
library(DataCombine)
rmExcept("train_test")
train = train_test[train_test$training_data ==1, ]
test = train_test[train_test$training_data ==2, ]
train$training_data = NULL
test$training_data = NULL
View(train)
library(C50)
install.packages("caret",repos = "http://cran.r-project.org", dependencies = c("Depends",
"Imports", "Suggests"))
c50_Model = C5.0(Churn~., train, trials = 100, rules = TRUE)
summary(c50_Model)
#write rules into disk
write(capture.output(summary(c50_Model)), "c50Rules.txt")
View(c50_Model)
#predict
c50_Predictions = predict(c50_Model, test[,-17], type = "class")
c50_Predictions

confMatrix_c50 = table(test$Churn, c50_Predictions)
library(caret)
confusionMatrix(confMatrix_c50)
confMatrix_c50

FPR = FN/FN+TP
```

#Accuracy = 95.74%

#28%

#Random Forest using 100 decision trees

```
library(randomForest)
library(RRF)
library(inTrees)
RF_Model = randomForest(Churn ~., train, importance = TRUE, nTree = 500)
```

```
#Extract rules from random forest
treelist = RF2List(RF_Model)

#Exctract Rules
exec = extractRules(treelist, train[,-17])

readableRules = presentRules(exec, colnames(train))
readableRules[1:2,]

ruleMetric = getRuleMetric(exec, train[,-17], train$Churn)
ruleMetric[1:2,]

RF_Predictions = predict(RF_Model, test[,-17])
confusionMatrix_RF = table(test$Churn,RF_Predictions)
library(caret)
confusionMatrix(confusionMatrix_RF)
```

##case = 100
##Accuracy = 94.84%
##FNR = 28.54%

#Case =500
##Accuracy =94.84%
##FNR = 30.35%

##Linear Regression cannot be used for classsification purpose and it can only be used as regression

##Logistic Regression:

```
logit_Model = glm(Churn~., data = train, family ="binomial")
summary(logit_Model)
logit_predictions = predict(logit_Model, newdata = test, type = "response")

logit_predictions = ifelse(logit_predictions >0.5, 1,0)
confusionMatrix = table(test$Churn, logit_predictions)
```

#Accuracy = 87.04
#FNR = 75.44

#KNN Model

```
library(class)
```

```
KNN_Predictions = Knn(train[,1:16], test[,1:16], train$target, k=1)


library(Rserve)
Rserve()
```

Box plot Visualizations for all the variables

Fig a

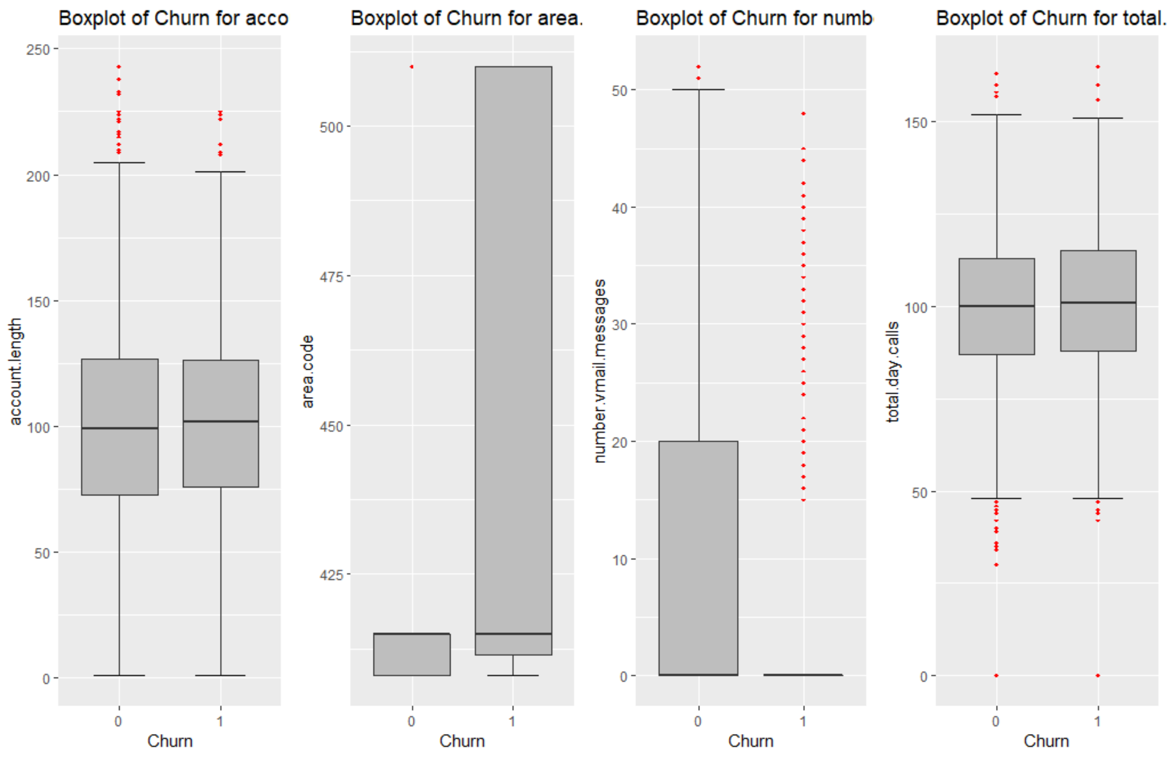


Fig b

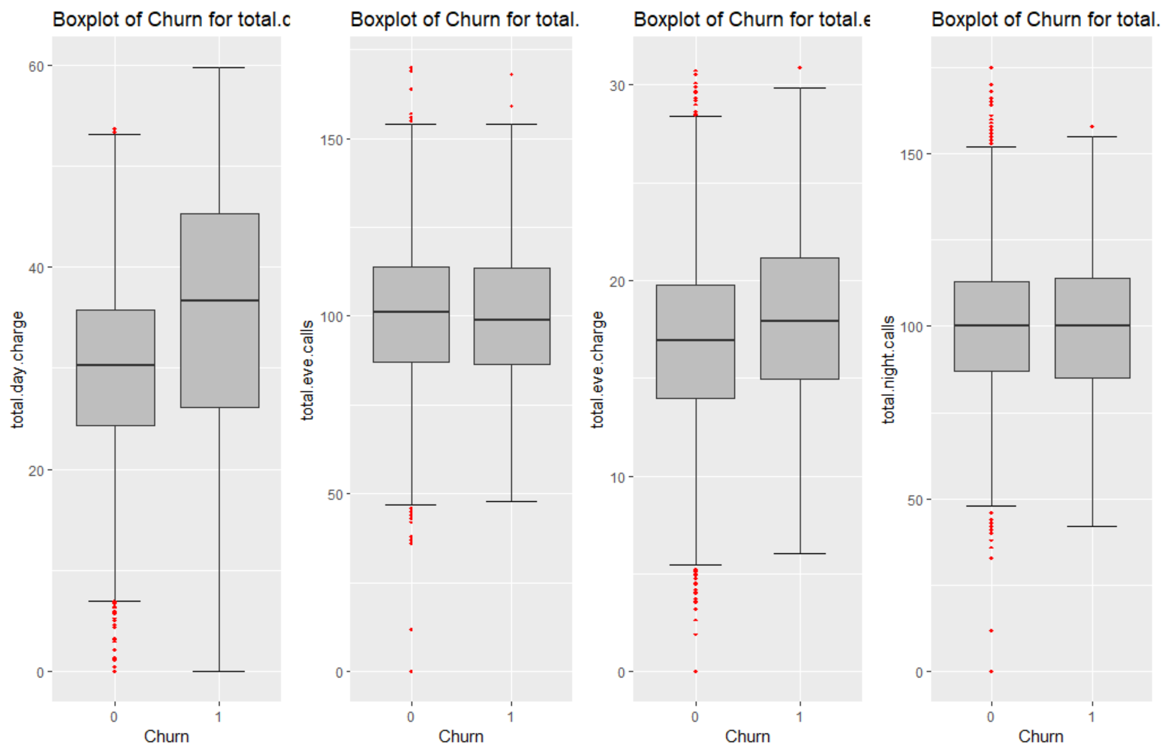

Boxplot of Churn for total.c
Boxplot of Churn for total.
Boxplot of Churn for total.e
Boxplot of Churn for total.
total.day.charge
total.eve.calls
total.eve.charge
total.night.calls
Churn

Fig c

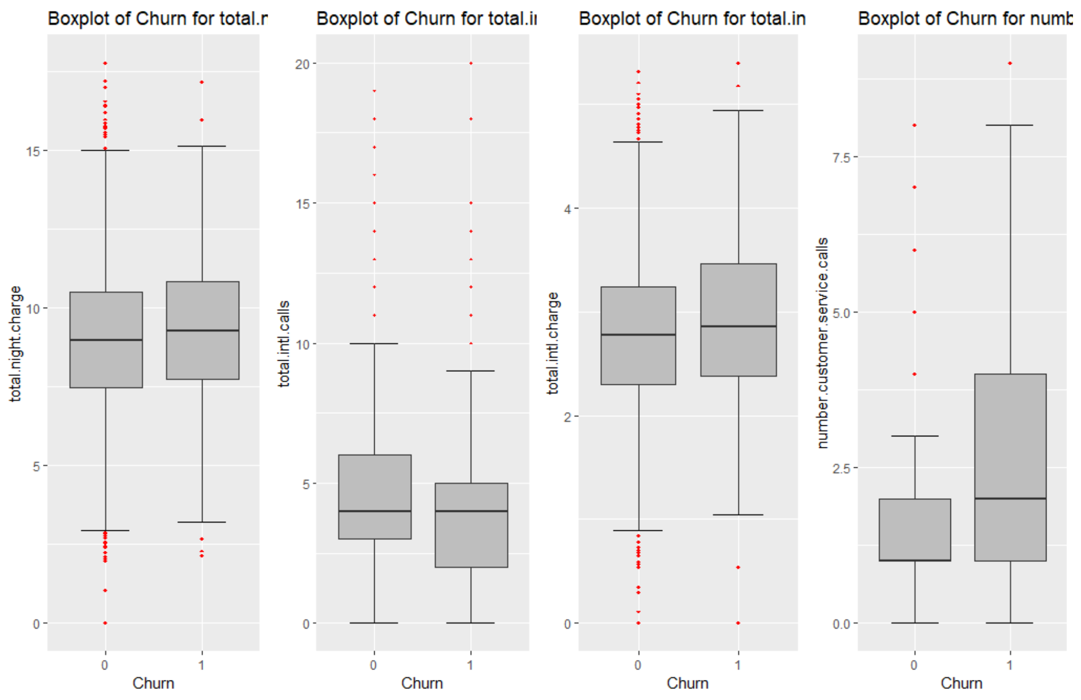

Boxplot of Churn for total.r
Boxplot of Churn for total.ii
Boxplot of Churn for total.in
Boxplot of Churn for numb
total.night.charge
total.intl.calls
total.intl.charge
number.customer.service.calls
Churn

Fig d

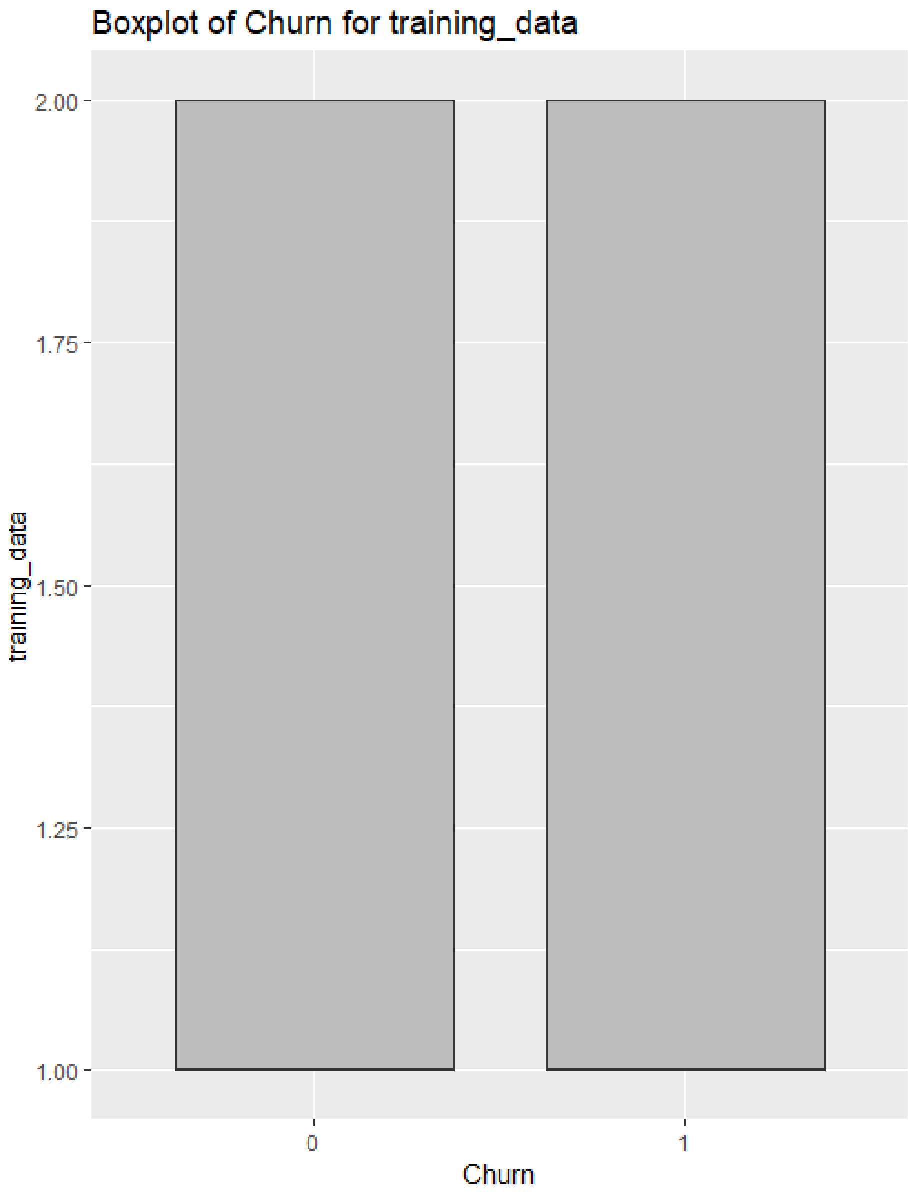
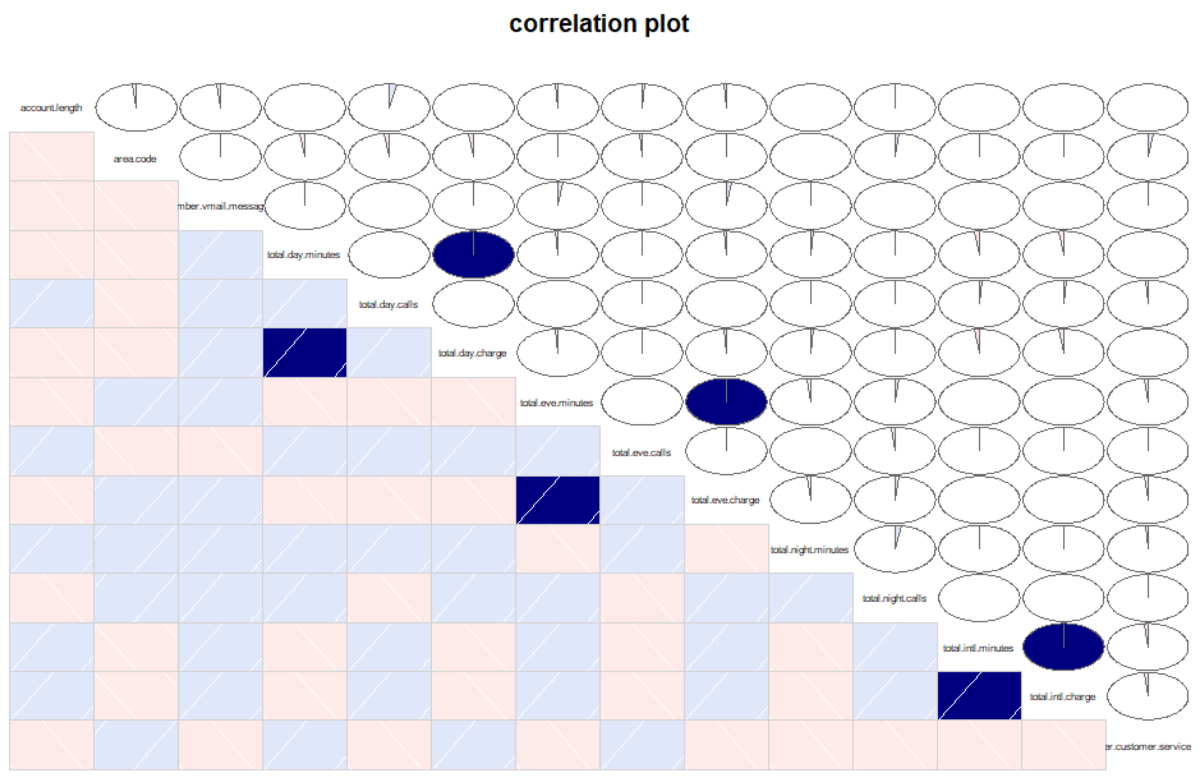

Boxplot of Churn for training_data
2.00
1.75
1.50
1.25
1.00
training_data
0
1
Churn

Fig e



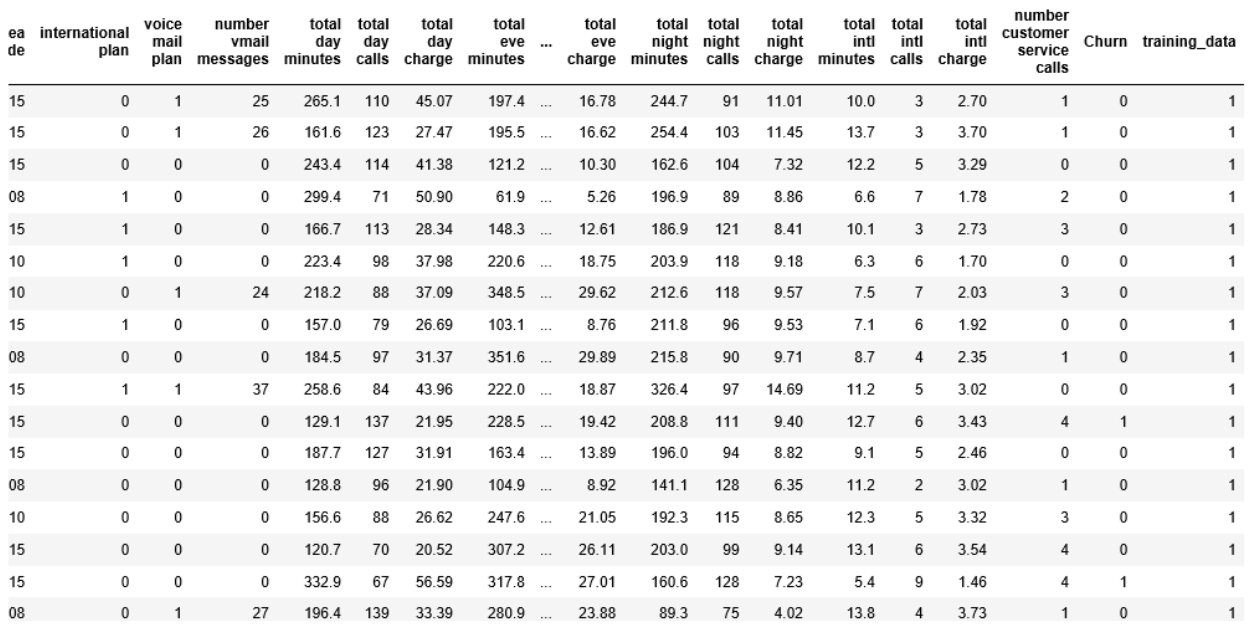
##Chi-square rules:

```
> for(i in 1:4)
+    {
+        print(names(factor_data)[i])
+        print(chisq.test(table(factor_data$Churn, factor_data[,i])))
+ }
[1] "state"

	Pearson's Chi-squared test

data:  table(factor_data$Churn, factor_data[, i])
X-squared = 96.899, df = 50, p-value = 7.851e-05

[1] "phone.number"

	Pearson's Chi-squared test

data:  table(factor_data$Churn, factor_data[, i])
X-squared = 5000, df = 4999, p-value = 0.4934

[1] "international.plan"

	Pearson's Chi-squared test with Yates' continuity correction

data:  table(factor_data$Churn, factor_data[, i])
```

```
X-squared = 333.19, df = 1, p-value < 2.2e-16

[1] "voice.mail.plan"

      Pearson's Chi-squared test with Yates' continuity correction

data:  table(factor_data$Churn, factor_data[, i])
X-squared = 60.552, df = 1, p-value = 7.165e-15

[1] "account.length"
[1] "area.code"
[1] "number.vmail.messages"
[1] "total.day.calls"
[1] "total.day.charge"
[1] "total.eve.calls"
[1] "total.eve.charge"
[1] "total.night.calls"
[1] "total.night.charge"
[1] "total.intl.calls"
[1] "total.intl.charge"
[1] "number.customer.service.calls"
[1] "training_data"
```

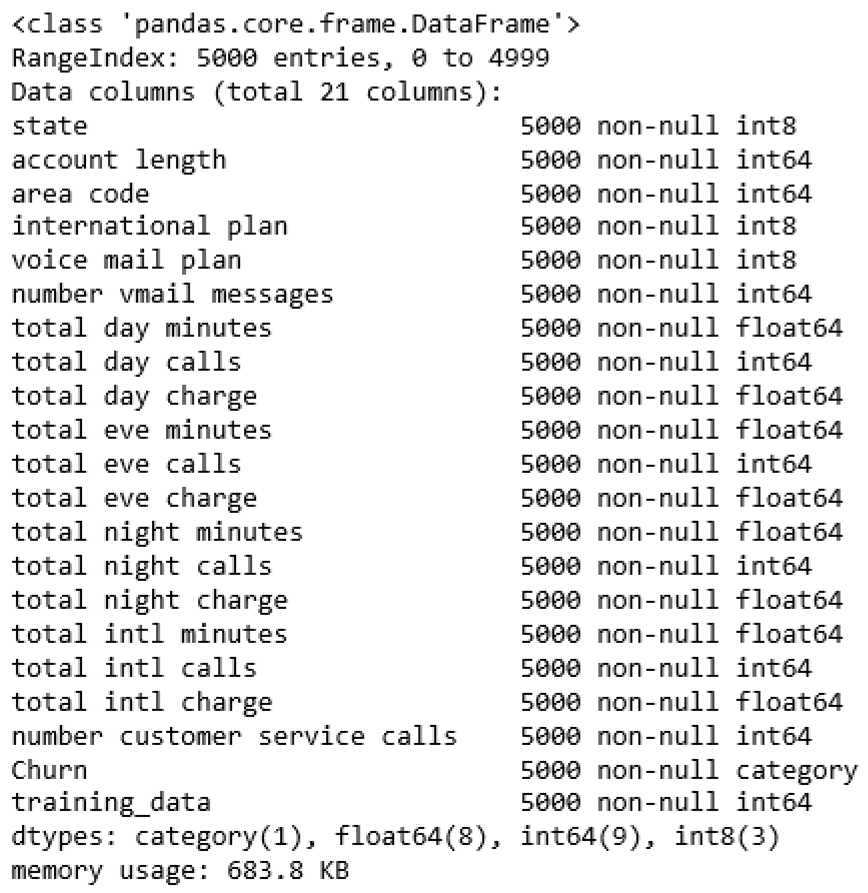
JAVA Code:

train_test.head(5000)



ea de	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	...	total eve charge	total night minutes	total night calls	total night charge	total intl minutes	total intl calls	total intl charge	number customer service calls	Churn	training_data
15	0	1	25	265.1	110	45.07	197.4	...	16.78	244.7	91	11.01	10.0	3	2.70	1	0	1
15	0	1	26	161.6	123	27.47	195.5	...	16.62	254.4	103	11.45	13.7	3	3.70	1	0	1
15	0	0	0	243.4	114	41.38	121.2	...	10.30	162.6	104	7.32	12.2	5	3.29	0	0	1
08	1	0	0	299.4	71	50.90	61.9	...	5.26	196.9	89	8.86	6.6	7	1.78	2	0	1
15	1	0	0	166.7	113	28.34	148.3	...	12.61	186.9	121	8.41	10.1	3	2.73	3	0	1
10	1	0	0	223.4	98	37.98	220.6	...	18.75	203.9	118	9.18	6.3	6	1.70	0	0	1
10	0	1	24	218.2	88	37.09	348.5	...	29.62	212.6	118	9.57	7.5	7	2.03	3	0	1
15	1	0	0	157.0	79	26.69	103.1	...	8.76	211.8	96	9.53	7.1	6	1.92	0	0	1
08	0	0	0	184.5	97	31.37	351.6	...	29.89	215.8	90	9.71	8.7	4	2.35	1	0	1
15	1	1	37	258.6	84	43.96	222.0	...	18.87	326.4	97	14.69	11.2	5	3.02	0	0	1
15	0	0	0	129.1	137	21.95	228.5	...	19.42	208.8	111	9.40	12.7	6	3.43	4	1	1
15	0	0	0	187.7	127	31.91	163.4	...	13.89	196.0	94	8.82	9.1	5	2.46	0	0	1
08	0	0	0	128.8	96	21.90	104.9	...	8.92	141.1	128	6.35	11.2	2	3.02	1	0	1
10	0	0	0	156.6	88	26.62	247.6	...	21.05	192.3	115	8.65	12.3	5	3.32	3	0	1
15	0	0	0	120.7	70	20.52	307.2	...	26.11	203.0	99	9.14	13.1	6	3.54	4	0	1
15	0	0	0	332.9	67	56.59	317.8	...	27.01	160.6	128	7.23	5.4	9	1.46	4	1	1
08	0	1	27	196.4	139	33.39	280.9	...	23.88	89.3	75	4.02	13.8	4	3.73	1	0	1

train_test.info()



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 21 columns):
state                            5000 non-null int8
account length                   5000 non-null int64
area code                        5000 non-null int64
international plan               5000 non-null int8
voice mail plan                  5000 non-null int8
number vmail messages            5000 non-null int64
total day minutes                5000 non-null float64
total day calls                  5000 non-null int64
total day charge                 5000 non-null float64
total eve minutes                5000 non-null float64
total eve calls                  5000 non-null int64
total eve charge                 5000 non-null float64
total night minutes              5000 non-null float64
total night calls                5000 non-null int64
total night charge               5000 non-null float64
total intl minutes               5000 non-null float64
total intl calls                 5000 non-null int64
total intl charge                5000 non-null float64
number customer service calls    5000 non-null int64
Churn                            5000 non-null category
training_data                    5000 non-null int64
dtypes: category(1), float64(8), int64(9), int8(3)
memory usage: 683.8 KB
```

```
p_values = []

for i in train_test.columns:

   print(i)

   chi2, p, dof, ex = chi2_contingency(pd.crosstab(train_test.Churn, train_test[i]))

   p_values.append(p)

   print(p)
```

```
state 7.850836224371827e-05 account length 0.9164350741820516 area code
0.7546581385329686 international plan 1.9443947474998577e-74 voice mail
plan 7.164501780988496e-15 number vmail messages 0.00020125744384152328
total day minutes 2.8394963850721003e-20 total day calls
0.07865300432636355 total day charge 2.8394963850721003e-20 total eve
minutes 0.06859434773011679 total eve calls 0.7612391081751979 total eve
charge 0.026866451343047174 total night minutes 0.8728894718024558 total
night calls 0.9405017867252645 total night charge 0.7547203743545484 total
intl minutes 0.0005006306730069841 total intl calls 1.3855418957748761e-05
```

```
total intl charge 0.0005006306730069841 number customer service calls
4.186291993492475e-101 Churn 0.0 training_data 0.3343167771249668
```

selected_features = []

for val in p_values:

    selected_features.append(val<0.05)

train_test = train_test.iloc[:, selected_features]

train_test.shape

```
(5000, 12)
```

train_test.head()

	state	international plan	voice mail plan	number vmail messages	total day minutes	total day charge	total eve charge	total intl minutes	total intl calls	total intl charge	number customer service calls	Churn
0	16	0	1	25	265.1	45.07	16.78	10.0	3	2.70	1	0
1	35	0	1	26	161.6	27.47	16.62	13.7	3	3.70	1	0
2	31	0	0	0	243.4	41.38	10.30	12.2	5	3.29	0	0
3	35	1	0	0	299.4	50.90	5.26	6.6	7	1.78	2	0
4	36	1	0	0	166.7	28.34	12.61	10.1	3	2.73	3	0

train_test = train_test.drop(labels=['total day minutes', 'total intl minutes'], axis=1)

train_test.shape

(5000, 10)

train_test.head()

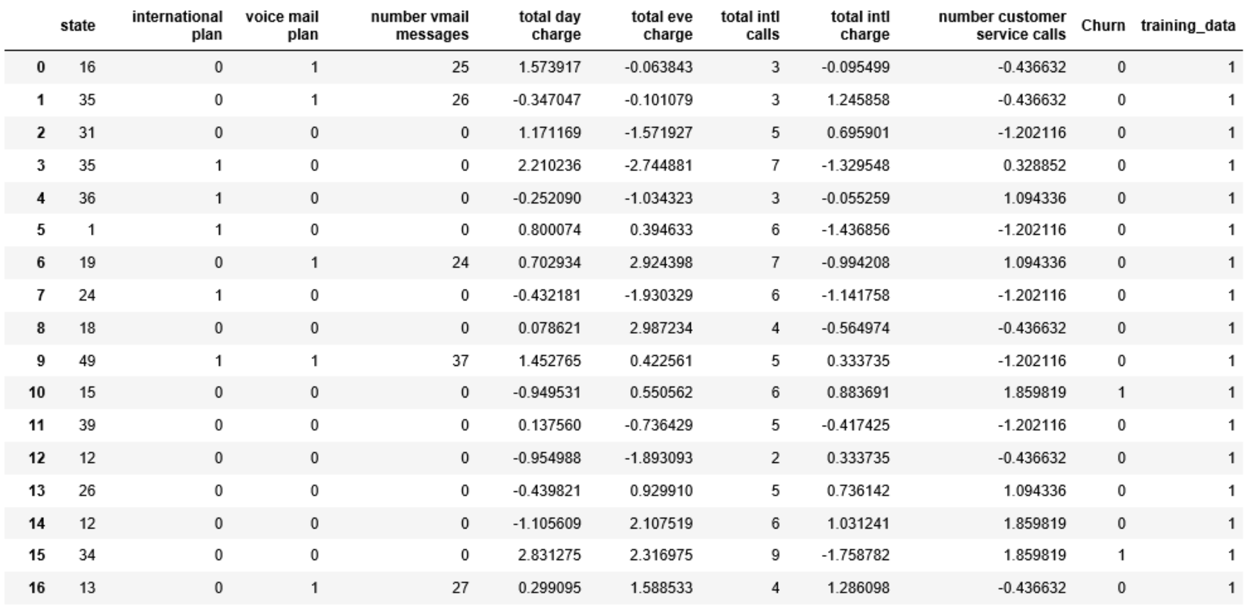
	state	international plan	voice mail plan	number vmail messages	total day charge	total eve charge	total intl calls	total intl charge	number customer service calls	Churn
0	16	0	1	25	45.07	16.78	3	2.70	1	0
1	35	0	1	26	27.47	16.62	3	3.70	1	0
2	31	0	0	0	41.38	10.30	5	3.29	0	0
3	35	1	0	0	50.90	5.26	7	1.78	2	0
4	36	1	0	0	28.34	12.61	3	2.73	3	0

cnames = ['total day charge','total eve charge', 'total intl charge', 'number customer service calls']

for i in cnames:

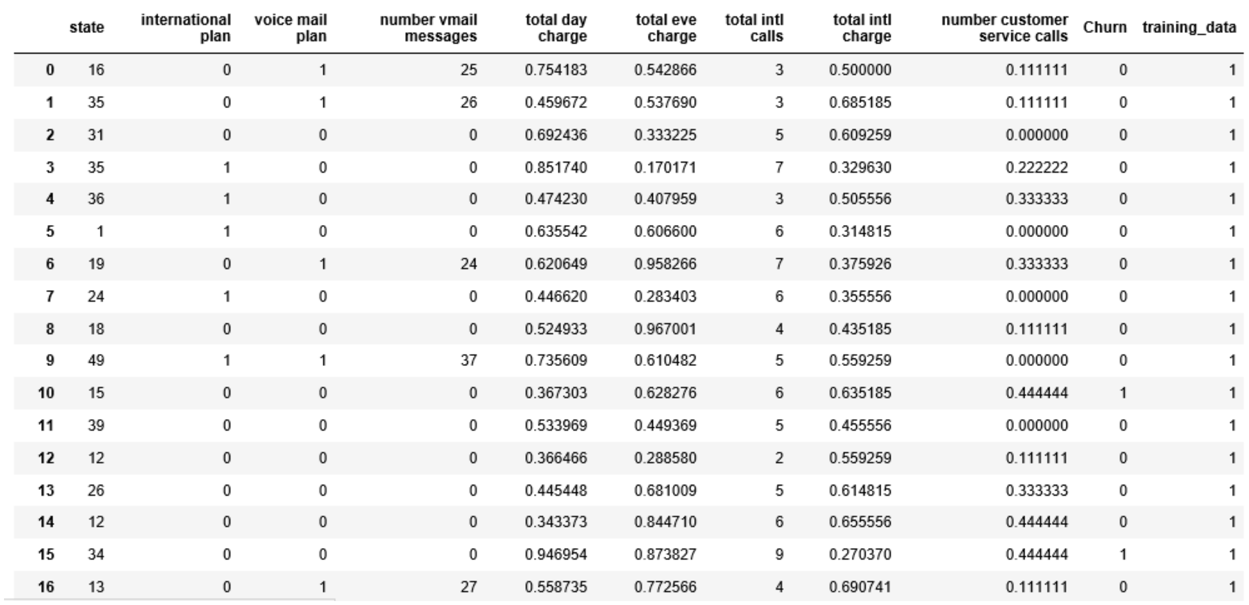
    print(i)

```
    train_test[i] = (train_test[i] - min(train_test[i]))/(max(train_test[i])-min(train_test[i]))

total day charge total eve charge total intl charge number customer service calls

for i in cnames:

    print(i)

    train_test[i] = (train_test[i] - (train_test[i]).mean())/train_test[i].std()

total day charge total eve charge total intl charge number customer service calls

train_test
```



	state	international plan	voice mail plan	number vmail messages	total day charge	total eve charge	total intl calls	total intl charge	number customer service calls	Churn	training_data
0	16	0	1	25	1.573917	-0.063843	3	-0.095499	-0.436632	0	1
1	35	0	1	26	-0.347047	-0.101079	3	1.245858	-0.436632	0	1
2	31	0	0	0	1.171169	-1.571927	5	0.695901	-1.202116	0	1
3	35	1	0	0	2.210236	-2.744881	7	-1.329548	0.328852	0	1
4	36	1	0	0	-0.252090	-1.034323	3	-0.055259	1.094336	0	1
5	1	1	0	0	0.800074	0.394633	6	-1.436856	-1.202116	0	1
6	19	0	1	24	0.702934	2.924398	7	-0.994208	1.094336	0	1
7	24	1	0	0	-0.432181	-1.930329	6	-1.141758	-1.202116	0	1
8	18	0	0	0	0.078621	2.987234	4	-0.564974	-0.436632	0	1
9	49	1	1	37	1.452765	0.422561	5	0.333735	-1.202116	0	1
10	15	0	0	0	-0.949531	0.550562	6	0.883691	1.859819	1	1
11	39	0	0	0	0.137560	-0.736429	5	-0.417425	-1.202116	0	1
12	12	0	0	0	-0.954988	-1.893093	2	0.333735	-0.436632	0	1
13	26	0	0	0	-0.439821	0.929910	5	0.736142	1.094336	0	1
14	12	0	0	0	-1.105609	2.107519	6	1.031241	1.859819	0	1
15	34	0	0	0	2.831275	2.316975	9	-1.758782	1.859819	1	1
16	13	0	1	27	0.299095	1.588533	4	1.286098	-0.436632	0	1

```
train_test['training_data'] = train_data.training_data

train_test.training_data = np.where(train_test.training_data==1, 1, 0)

train = train_test[train_test.training_data == 1]

test = train_test[train_test.training_data == 0]

train = train.drop(labels='training_data', axis=1)

test = test.drop(labels='training_data', axis=1)

train_test
```

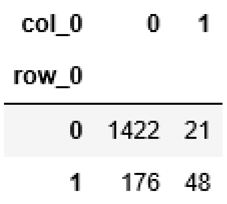


	state	international plan	voice mail plan	number vmail messages	total day charge	total eve charge	total intl calls	total intl charge	number customer service calls	Churn	training_data
0	16	0	1	25	0.754183	0.542866	3	0.500000	0.111111	0	1
1	35	0	1	26	0.459672	0.537690	3	0.685185	0.111111	0	1
2	31	0	0	0	0.692436	0.333225	5	0.609259	0.000000	0	1
3	35	1	0	0	0.851740	0.170171	7	0.329630	0.222222	0	1
4	36	1	0	0	0.474230	0.407959	3	0.505556	0.333333	0	1
5	1	1	0	0	0.635542	0.606600	6	0.314815	0.000000	0	1
6	19	0	1	24	0.620649	0.958266	7	0.375926	0.333333	0	1
7	24	1	0	0	0.446620	0.283403	6	0.355556	0.000000	0	1
8	18	0	0	0	0.524933	0.967001	4	0.435185	0.111111	0	1
9	49	1	1	37	0.735609	0.610482	5	0.559259	0.000000	0	1
10	15	0	0	0	0.367303	0.628276	6	0.635185	0.444444	1	1
11	39	0	0	0	0.533969	0.449369	5	0.455556	0.000000	0	1
12	12	0	0	0	0.366466	0.288580	2	0.559259	0.111111	0	1
13	26	0	0	0	0.445448	0.681009	5	0.614815	0.333333	0	1
14	12	0	0	0	0.343373	0.844710	6	0.655556	0.444444	0	1
15	34	0	0	0	0.946954	0.873827	9	0.270370	0.444444	1	1
16	13	0	1	27	0.558735	0.772566	4	0.690741	0.111111	0	1

```
print("Shape of train : " + str(train.shape))

print("Shape of test : " + str(test.shape))

Shape of train : (3333, 10) Shape of test : (1667, 10)

train = train.drop(labels='state', axis=1)

test = test.drop(labels='state', axis=1)

x_train = train.values[:,0:8]

x_train

y_train = train.values[:,8]

y_train

x_test = test.values[:,0:8]

x_test

y_test = test.values[:,8]

y_test

clf = tree.DecisionTreeClassifier(criterion='entropy').fit(x_train.astype('int'),y_train.astype('int'))

clf
```

```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=None, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, presort=False, random_state=None, splitter='best')
```

```
y_predict = clf.predict(x_test).astype('int')

y_predict

accuracy_score(y_test.astype('int'), y_predict)*100

from sklearn.metrics import confusion_matrix

CM = confusion_matrix(y_test.astype('int'), y_predict.astype('int'))

CM = pd.crosstab(y_test.astype('int'), y_predict.astype('int'))

TN = CM.iloc[0,0]

FN = CM.iloc[1,0]

TP = CM.iloc[1,1]

FP = CM.iloc[0,1]

CM
```



```
 col_0     0   1
 row_0
     0  1422  21
     1   176  48
```

#accuracy

((TP+TN)*100)/(TP+TN+FP+FN)

```
88.18236352729454
```

accuracy_score(y_test.astype('int'), y_predict)*100

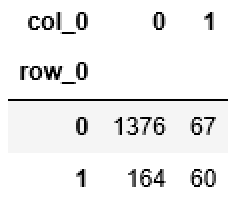
```
88.18236352729454
```

#FNR---its quite high

(FN*100)/(FN+TP)

```
78.57142857142857
```

```
#Random Forest Classifier

from sklearn.ensemble import RandomForestClassifier

RF_Model = RandomForestClassifier(n_estimators = 500).fit(x_train.astype('int'),y_train.astype('int'))

RF_Predictions = RF_Model.predict(x_test)

CMR = pd.crosstab(y_test.astype('int'), RF_Predictions )

CMR
```



col_0	0	1
row_0		
0	1376	67
1	164	60

```
TN = CMR.iloc[0,0]

FN = CMR.iloc[1,0]

TP = CMR.iloc[1,1]

FP = CMR.iloc[0,1]

#accuracy

((TP+TN)*100)/(TP+TN+FP+FN)
```

```
86.14277144571086
```

```
(FN*100)/(FN+TP)
```

```
73.21428571428571
```

```
from sklearn.neighbors import KNeighborsClassifier

KNN_Model = KNeighborsClassifier(n_neighbors = 1).fit(x_train.astype('int'),y_train.astype('int'))

KNN_predictions = KNN_Model.predict(x_test.astype('int'))

CMK = pd.crosstab(y_test.astype('int'),KNN_predictions)
```

```
TN = CMK.iloc[0,0]

FN = CMK.iloc[1,0]

TP = CMK.iloc[1,1]

FP = CMK.iloc[0,1]
```

((TP+TN)*100)/(TP+TN+FP+FN)

```
84.98273878020713
```

(FN*100)/(FN+TP)

```
86.82170542635659
```

```
from sklearn.naive_bayes import GaussianNB

NB_model = GaussianNB().fit(x_train.astype('int'),y_train.astype('int'))

NB_predictions = NB_model.predict(x_test)

CMN = pd.crosstab(y_test.astype('int'), NB_predictions)

TN = CMN.iloc[0,0]

FN = CMN.iloc[1,0]
```

5.0.0 Conclusion

By comparing all the models and their accuracy we found that in decision tree model is having 95% accuracy in R Model and in Python we can lock Random Forest Classifier is having 86% accuracy.